Universitat de Lleida

TREBALL FINAL DE MÀSTER

Estudiant: **Carolina Romeu Farré**

Titulació: Màster en Enginyeria Informàtica

Títol de Treball Final de Màster: Design and implementation BLAST tool big data

Director/a: Fernando Cores Prado

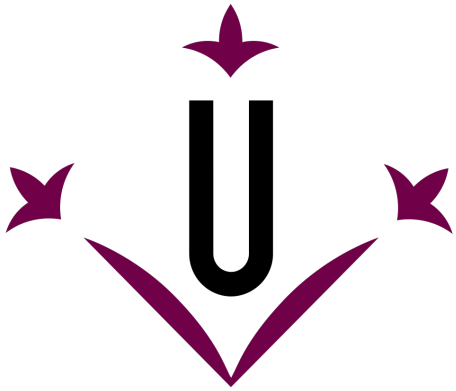
Presentació

Mes: Setembre

Any: 2019

Master thesis on Computational Engineering

Universitat de LLeida



Universitat de Lleida

Design and implementation BLAST tool big data

Carolina Romeu Farré

Supervisor: Fernando Cores Prado

September 2019

Contents

Acknowledgement

After this long time developing this project, I would like to thank the people who have made it possible. Especially:

- To my project supervisor Fernando Cores.
- To my partner Soledad Moraga.

Chapter 1

Introduction

The large volume of data that is generated today in all fields of industry and science is revolutionizing the way how to interact with applications [1].

As the data systems are updated, the various tools for processing do so as well. Covering optimization and usability improvements that facilitate their learning and integration. However, despite the tireless improvement and updating of the tools, it is intuited that there will be an exponential growth of data in the different areas. Which entails new optimization challenges that allow to ensure the processing of large volumes of data, at an optimal speed ensuring its consistency, integrity and variety.

All these large volumes are widely used in commerce and medicine, they can be found in different applications in law, sociology, advertising, health care and in all areas of natural sciences. The data, in all its forms, have the potential to provide a useful flow of information whenever the way to extract it is developed. The new techniques combine traditional statistics with computer science and make the analysis of large volumes of data more viable [2].

Now, the theme has been converted in fashion and that it has ceased to be associated only with the large volume of data. There are also possibilities to explore the application of Big Data to new data domains, since they have currently focused on

social media, medicine, bioinformatics and security, mainly [3].

For example, the volume of data generated by research in Bioinformatics is very high and highly heterogeneous in nature. Big data sources are no longer limited to particle physics experiments or search-engine logs and indexes. Whereby, many analytics problems in bioinformatics require multiple heterogeneous and independent databases for inference and validation.

The European Bioinformatics Institute (EBI), one of the largest biology-data repositories, had approximately 40 petabytes of data about genes, proteins, and small molecules in 2014, in comparison to 18 petabytes in 2013. There are many other organizations, who are storing and processing huge collections of biological databases and distributing them around the world, such as National Center for Biotechnology Information (NCBI), USA and National Institute of Genetics, Japan [4].

The all technological development acquired since the decipherment of the human genome allows some of this knowledge to be applied to the health system. The bioinformatics is responsible for analyzing this large volume of data and interpret them and give them a sense that allows their correct application, so that the doctor or other health care professional can make better decisions in the management and treatment of their patients [5] or the application of called individualized medicine.

Personalized medicine

The term "personalized medicine" as outlined is often used particularly in connection with pharmacotherapy/bio-markers and/or "genetic therapy" [6].

As you can see in the figure 1, the context in which individualized medicine would be developed is gradual and requires greater specialization of genetic processes. Starting with a generalized medicine, moving towards a personalized medicine that encompasses biomarkers that we can currently find in very special medical procedures until reaching an individualized therapy, which will require a level of specialization in the study and very high genomic prediction.

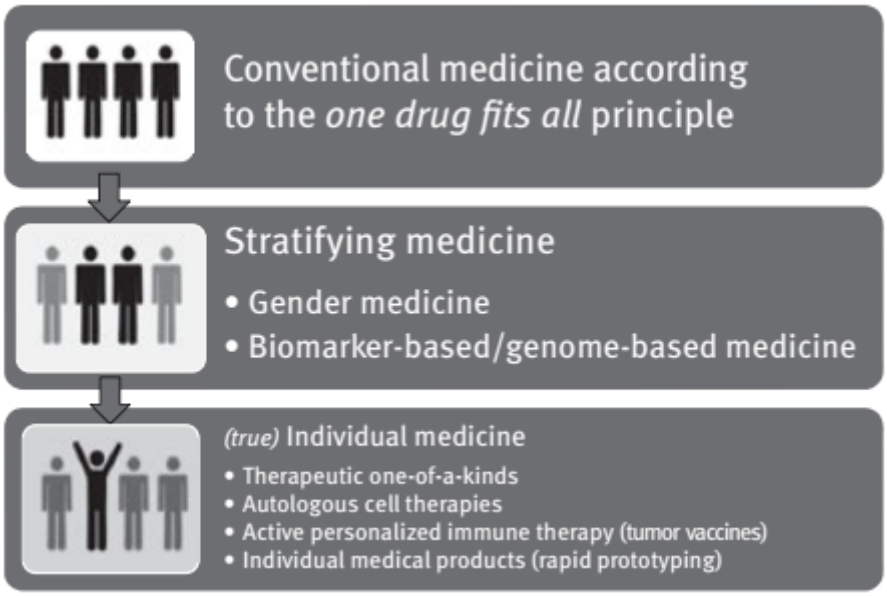


Figure 1: The path to "personalized medicine"

To provide each patient with an individual therapy that acts painlessly, quickly and easily for some that is a vision, for many others it might sound delusional. Often personalized medicine, particularly in the context of Big Data, is understood to mean the connection of genetic code and possible healing options. Hardly any other area of medicine has seen such an increase ("explosion") of data in the last five years, with the corresponding faster and cheaper analysis of the data. Therefore, the hype about Big Data and genetics can also be explained through these new developments [6].

The human genome (genomic DNA) has a length of 2 x 3.2 Mb with approximately 19,000 to 26,000 protein-coding genes, 1500 genes for micro-RNAs, some 8900 pseudogenes and approximately 290 disease genes for monogenic (Mendelian) genetic diseases. That said, the large amount of data that can be obtained from a genome and genetic application is very wide and variant [4].

Therefore, as gene expression data are being captured at different progression stages of a disease over time, there has been an opportunity to identify the genes that are affected by the disease, in order to identify biomarkers for the disease. Computationally, the addition of the third dimension, time, makes the analytics much higher in complexity than the traditional analysis of gene complexes. Gene co-expression network analysis is a complex and highly iterative problem and requires large-scale data analytics systems [4].

So, with the rapid growth of biological sequence datasets and the evolution of the sequencing technologies, many algorithms and software systems commonly used for the analysis of biological sequences are becoming obsolete. For this reason, computational approaches based on frameworks for big data processing started to be proposed in order to deal with problems involving large amounts of biological data [7].

Bigdata and problems with bioinformatics

- As gene expression data are being captured at different progression stages of a disease over time, there has been an opportunity to identify the genes that are affected by the disease, in order to identify biomarkers for the disease. Computationally, **the addition of the third dimension, time, makes the analytics much higher in complexity than the traditional analysis of gene complexes** [4].

- Gene co-expression network analysis is a complex and highly iterative problem and requires large-scale data analytics systems [4].

- With the increasing volume (in order of petabytes) of DNA data deluge origi-

nated from thousands of sources, the present DNA sequencing tools have been found inadequate [4].

1.1 Context

The sequencing of complete genomes is a powerful method for the rapid identification of genes in an organism, and serves as a basic tool for subsequent functional analyzes of the new genes discovered. The genomic sequence provides a set of virtually all the proteins that the body can express [8].

After all the sequencing processes of a successful genome assembly, the next challenge is to interpret the information it contains. This requires the identification of the main characteristics of the genome, a process known as annotation [8]. Where two important processes can be identified: Structural annotation (prediction of coding regions) and functional annotation (allocation of biological information to previously predicted genes).

And based on the premise that **genes with shared sequences also share their function**. The function of a gene can be inferred by searching for homologous sequences in databases, using local alignment algorithms, such as BLAST [8]. Which was developed as a way to perform similarity searches of DNA sequences and proteins using an algorithm. A heuristic method where short common patterns are found in query and database sequences and join them in an alignment.

The BLAST algorithm has evolved to provide molecular biologists with a set of very powerful search tools that are freely available to run on many computing platforms [9]. To execute, BLAST requires two sequences as input: a query sequence (also called a white sequence) and a sequence database. BLAST specifies sub-sequences in the query that are similar to sub-sequences in the database. In typical use, the query sequence is much smaller than the database, for example, the query can be one thousand nucleotides while the database is several billion nucleotides [10].

In addition, BLAST is very popular due to its availability on the World Wide Web through a large server at the National Center for Biotechnology Information (NCBI) and many other sites. Currently NCBI BLAST, can be run via web, entering the

sequence you want to analyze and the database you want to use. In the figure 2, an example of execution of BLAST is shown from the NCBI website, in which it shows the following results, where the parts within the matching genome can be observed.

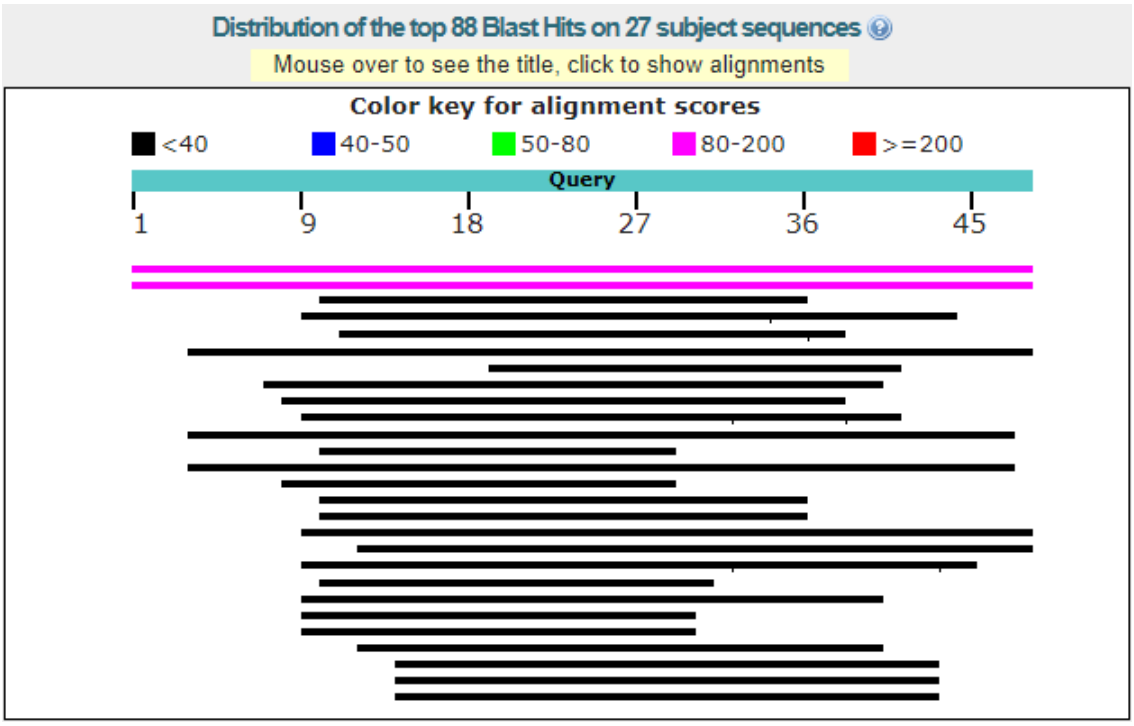


Figure 2: Results of NCBI BLAST on web site

As stated above, it is at this point that we found the importance of this project, which consists in optimizing the genome sequence, its handling and storage using BLAST and Apache Spark for a more efficient execution of the processes in a Big Data infrastructure.

Because maintaining the reference genomes for all the species analyzed and being able to process the queries in a single machine can generate problems of loading and saturation, in addition to not being a scalable solution. Another alternative to support a large volume of applications against a multitude of reference genomes is to be treated independently. But this alternative is clearly inefficient and very expensive at the hardware level since different requests / genomes do not share information (indexes, reference genome).

For this reason, the idea of the project is to be able to manage the sequences, use a distributed system for storage and execution, where BLAST will be reimplemented using the paradigms for massive data processing. Allowing the reuse of information, improving the efficiency and scalability of the tool. Thanks to the use of distributed infrastructure for massive data processing, such as Apache Hadoop or Spark. Proposing to save the different hashes of reference genomes in a distributed BD (Cassandra) so that they can be shared and reused. At the same time, a distributed framework is used to process the queries.

1.2 Objectives

- The main objective is to carry out a program that is capable of aligning the genetic sequences, using the BLAST algorithm and adapting it to the Big-Data distributed computing tools.

Consequently, it encompasses the following objectives:

- Analyze the operation of the BLAST program.
- Design and implement the program in Spark, using the distributed tools available.
- Evaluate implementation performance using large amounts of genetic sequence data.
- Obtain conclusions from the performance tests.

1.3 Planning and task

This section is dedicated to the structuring of the different tasks and activities involved in the study and development of this project.

1.3.1 Analysis

- **Investigate original BLAST performance**

A search has been made for information related to the **Blast** program, and on the operation of the algorithm that it uses to align the sequences.

- **Technologies involved**

  Search of the technologies that the project would imply and how to use them.

1.3.2 Implementation

- **Preparation environments**

  At this point, the workspace has been prepared and different tools that were needed as the implementation progressed

- **First part implementation. Create reference library** At this point, the first part of the implementation has been developed, which consists of saving the references of a genome in the Cassandra database.

- **Second part implementation. Process the queries.**

  It corresponds to the implementation of processing the sequence introduced to the program.

- **Implementation third part. Align sequence**

  In the final process of the implementation, the sequences resulting from the two previous processes have been aligned.

1.3.3 Testing

- **Program execution with different genomes and sizes.**

- **Test cluster performance.**

1.3.4 Documentation

- **Memories**

The figure 3 shows the initial planning of the project, defining like to deadline the month of July, so the planning shown represents the stages in a range of eight

months. If it is not possible delivery on this date, or see it convenient to postpone delivery, said date will be postponed until September.

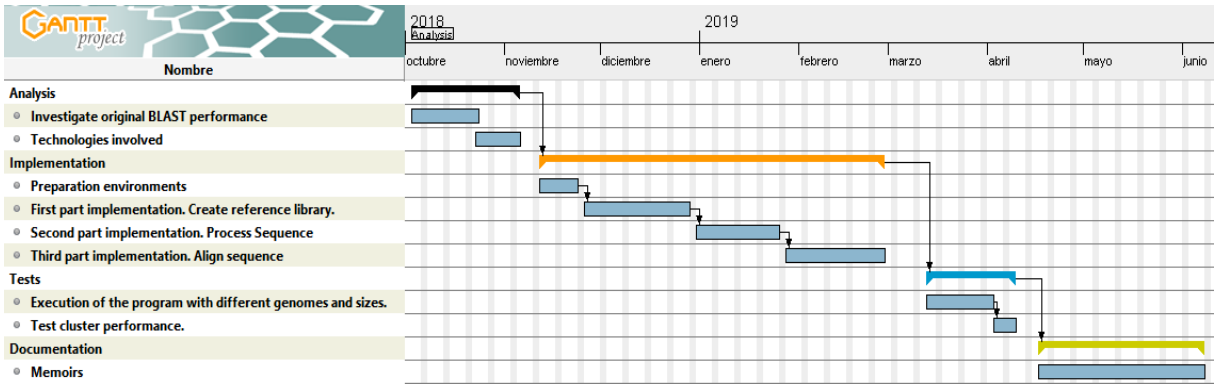


Figure 3: Project Gantt chart

1.4 Budget

- **Licenses**: All libraries and software used in this project are Open Source, so it does not imply any cost associated with its use.

- **Personnel**: In terms of personnel, a single role has been designated for this estimate, which is divided into hours worked differently according to the days of the week and weekends. In addition, the average hours worked per day does not exceed two hours per day. All corresponding holidays have also been removed.

Type of cost	Hours	Price	Total
Research costs	76	20 €	1.520€
Development costs	0	20 €	0 €
System implementation costs	198	20 €	3.960 €
Testing costs	56	20 €	1.120 €
Implementation Environment (Cluster)	10	20 €	200 €
		Total	6.800 €

Table 1: Breakdown of project costs.

1.5 Structure of the Report

This document is organized in six chapters, the content of which is described below.

The **chapter 1 (Introduction)** describes the theme of the project, introducing the current state of the use of massive data and its use in medicine and the study of genomes; likewise a structuring of the development of the project, planning and approximate costs is made. The objectives to be achieved are also presented.

The **chapter 2 (State of the art)** presents a bibliographic review of the main themes on which the project proposal is based: big data and its importance in the current technological development and necessary algorithms. The chapter concludes with the study of the technologies to be used.

The **chapter 3 (Analysis and Design)** presents the definition of the different requirements and the determination of tools, environment and programming languages to be used for the development of the project.

The **chapter 4 (Implementation)** explains in detail the development of the programs. Also the decisions taken throughout the implementation, the problems encountered and the solutions applied.

In **chapter 5 (Results and validation)** the results obtained in the execution of the program are detailed, then tests are carried out. Finally, the results obtained are recorded and later concluded.

The **chapter 6 (Conclusions)** presents the conclusions of the project, contributions and recommendations for future research in the same thematic proposed by the thesis.

Chapter 2

State-of-art

In this section, the technologies involved in the development of the project will be described. First, a more extensive explanation of what is BLAST program is going to be introduced. Subsequently the technologies involved throughout the implementation.

2.1 Basic Local Alignment Search Tool

BLAST (Basic Local Alignment Search Tool) has become the standard in search and alignment tools. The BLAST algorithm is still actively being developed and is one of the most cited papers ever written in this field of biology. Many researchers use BLAST as an initial screening of their sequence data from the laboratory and to get an idea of what they are working on. BLAST is far from being basic as the name indicates; it is a highly advanced algorithm which has become very popular due to availability, speed, and accuracy [11].

The BLAST algorithm and program were designed by Stephen Altschul, Warren Gish, Webb Miller, Eugene Myers, and David J. Lipman at the National Institutes of Health and was published in the Journal of Molecular Biology in 1990.

The local method uses a subset of a sequence and attempts to align it to subset of other sequences. Local alignments reveal regions that are highly similar but do not necessarily provide a comparison across the entire two sequences.

2.1.1 Algorithm

The algorithm uses the heuristics based on Smith-Waterman algorithm but improved by the aforementioned designers.

Local alignments use heuristic programming methods that are best suited to the successful search of very large databases, but do not necessarily offer the optimal solution. Even considering this limitation, local alignments are very important for the field of genomics because they can discover regions of homology that are related by descent between two different sequences.

BLAST is a set of algorithms that attempt to find a short fragment of a query sequence that perfectly aligns with a fragment of a subject sequence that is found in a database.

That initial alignment must be greater than a neighborhood score threshold (T). For the original BLAST algorithm, the fragment is then used as a seed to extend the alignment in both directions.

The alignment is extended in both directions until the T score for the aligned segment does not continue to increase. Said another way, BLAST looks for short sequences in the query that matches short sequences found in the database.

The alignment process is the one that will discover the most similar areas. The operation of the algorithm would be as follows:

The first step of the BLAST algorithm is to break the query into short words of a specific length. A word is a series of characters from the query sequences. The default length of the search is three characters. In this project we have used a longer length to make searches faster. Words are constructed using a sliding window of N characters.

In this case 3-character sub-sequences have been used as can be seen in figure 4, we will call it Query:

NYLENFVQATFN

NYL YLE LEN ENF NFV FVQ VQA QAT ATF TFN

Figure 4: Example of sequence

These words are compared with a sequence in a database, figure 5:

Query ENF

Subject SSTNYAENTIQSIISTVEPAQR

Figure 5: Example of sub-sequence and its comparison

This search is performed for all words. For the original BLAST search, those words whose T value was greater than 18 were used as seeds to extend the alignment.

The T value is derived by using a scoring matrix. The BLOSUM 62 matrix is the default for protein searches and will be discussed later. The alignment is extended in both directions until the alignment score decreases in value. For example:

Query NLYENFVQATFNALTAEKV

NY ENF+Q+++++

Subject NYAENTIQSIISTVEPAQR

Figure 6: Example of Alignment score

2.2 Technologies

In this section, it will do a bibliographic review of the different tools that will be used in the project, highlighting their characteristics and main uses. Basing the importance of your choice.

2.2.1 Big Data

Today, data is the aspiration of any company. The value they have acquired with technology, and the large volume that is used to solve problems, improve well-being, decision making and even economic predictions.

Data collection, storage and analysis are on an upward path[12] and seemingly in unlimited and exponential growth,which obviously has been driven by improvements in processing, improvements in data management techniques and especially in the optimization of calculations that facilitates interaction with this data.

In 2011, some estimated the amount of information created and replicated would surpass 1.8 zettabytes. 4 In 2013, estimates reached 4 zetta bytes of data generated worldwide.[12]

$$1 \text{ zettabyte} = 1{,}000{,}000{,}000{,}000{,}000{,}000{,}000 \text{ bytes}$$

There are many ways to define Big-Data, since these vary depending on who does it. However, many of these converge by reflecting the increasing technological capacity to capture, aggregate and process an increasing volume, speed and variety of data. Which is an abstract concept that in recent years has become a buzzword in different fields: business, computer engineering, information and documentation, or information systems, among others [13].

The term big data was coined by Doug Laney in the early 2000s. Laney's definition includes three concepts:

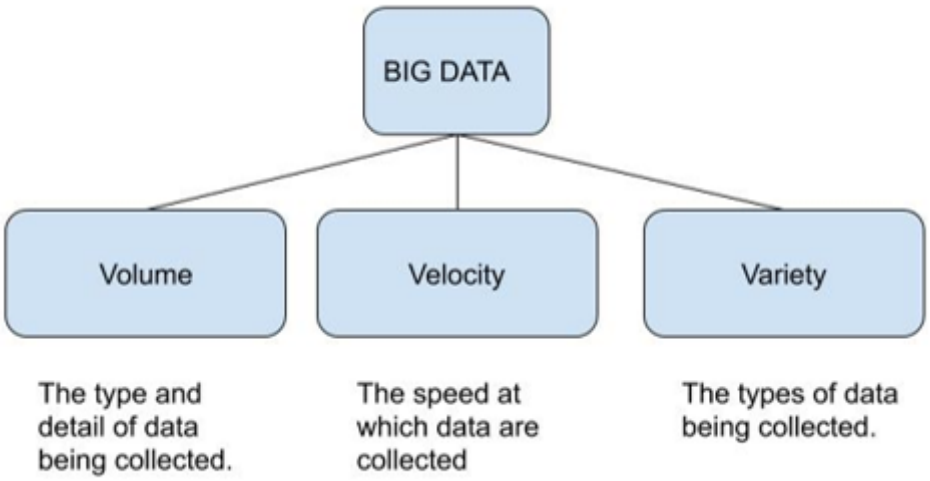


Figure 7: Conceptual map of big data

Since Laney's original work, another concept has been added: veracity. This describes how much "noise" is in the data. Excessively large amounts of data can make it difficult to identify which data are important and which data are distractions.

However, several more concepts can be identified as an annex to the original principles, for example:

- **Value**: knowledge value and information for decision making.
- **Display**:This feature has to do with the benefits of the software that manages the data. This should make it easier for large volumes of data to be presented visually in a practical, dynamic, interactive and understandable way [13].
- **Viability**: This concept refers to the attention to be paid to the infrastructure and the cost of working with Big-Data.

Life cycle of big data

The life cycle of big data can be divided into four phases:

- Collection: It is all data collection, from all possible sources.
- Compilation and consolidation: It refers to all the entities that gather all the data that is found, stored and processed.
- Data mining and analytics: One form of analytics is descriptive—the objective is to uncover and summarize patterns or features that exist in data sets. By contrast, predictive data analytics refers to the use of statistical models to generate new data. Developing and testing the models that find patterns and make predictions can require the collection and use of copious amounts of data. In a market context, a common purpose of big data analytics is to draw inferences about consumers' likely choices.
- And use: It emphasizes the use and how the use of mass data is. For example: How companies could use big data to help consumers and the steps they can take to avoid damaging consumers inadvertently through analysis.

2.2.2 Apache Hadoop

Hadoop was created by Doug Cutting, the creator of Apache Lucene, the widely used text search library. Hadoop has its origins in Apache Nutch, an open source web search engine, itself a part of the Lucene project.

The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures [14]. On the other hand, Hadoop also stands out for having an architecture capable of ensuring high availability and recovery of the data it consumes.

The hadoop architecture can be seen represented in the figure 8. A Hadoop cluster is a special type of computational cluster designed specifically for storing and analyzing huge amounts of unstructured data in a distributed computing environment. To meet the challenge of the large size of files in Big Data, HDFS "breaks" these files into blocks (configurable size, although they are usually 128MB or 256MB), and then distributes them among the different Data Nodes that make up the HDFS cluster.

In addition to distributing the blocks between different data nodes, it also replicates them (at least in 3 different nodes, 2 in the same rack and 1 in another) to avoid loss of information if any of the nodes fail [15], this means that Hadoop is based on a Master / Slave architecture with types of nodes: master node (master) and slave nodes (slaves). A Hadoop cluster has a single master node and several slave nodes.

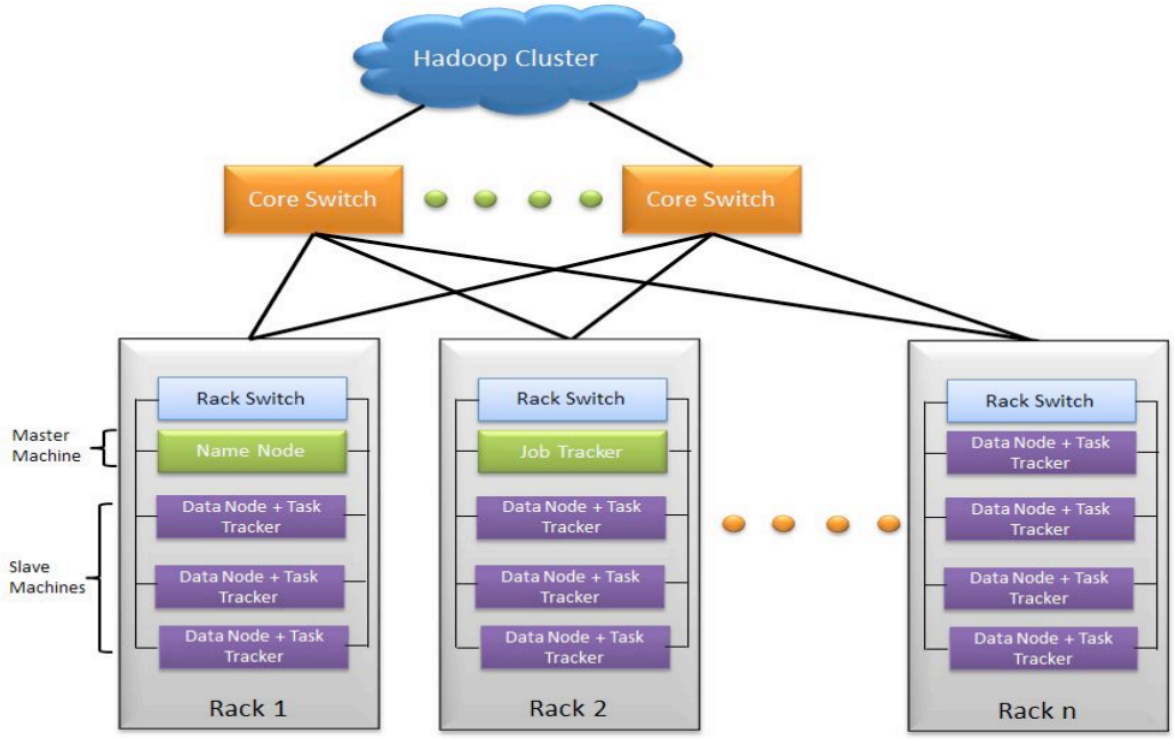


Figure 8: Hadoop cluster schema

2.2.3 Apache Spark

The Spark project was initiated by Matei Zaharia at the University of California at Berkeley in 2009 as a work of his doctoral thesis, and became open source in 2010. Since its creation, Spark has been a very active project, with a growing number of contributors (500 collaborators from more than 200 organizations) in each new version. The very active community continues to develop and publish updates Spark [1].

Apache Spark is an open-source unified big-data processing framework built around speed, ease of use and sophistical analytics. It is a general-purpose cluster computing framework with language-integrated APIs in Scala, Java, Python and R [16]. It is a platform to improve the performance of big data. This platform was born to make answers more quickly and efficiently, mainly interactive activities and iterative algorithms. However, performance of a particular job on Apache Spark platform can vary significantly depending on the input data type and size, design and implementation of the algorithm, and computing capability, making it extremely difficult to predict the performance metric of a job such as execution time, memory footprint, and I/O cost [17].

Spark can be seen represented in the figure 9 as an ecosystem that is made up of multiple tools and technologies. Since Spark is different from all the other open source projects in that it has numerous projects including but not limited to core Apache Spark runtime, Spark SQL, Spark Streaming, MLlib, ML, and GraphX and so on. In addition, outside of the foundational Spark libraries, there is an expanding ecosystem of other Spark libraries including but not limited to Zeppelin, SparkR, SnappyData, CaffeOnSpark, Spark Cassandra, BlinkDB and Tachyon.

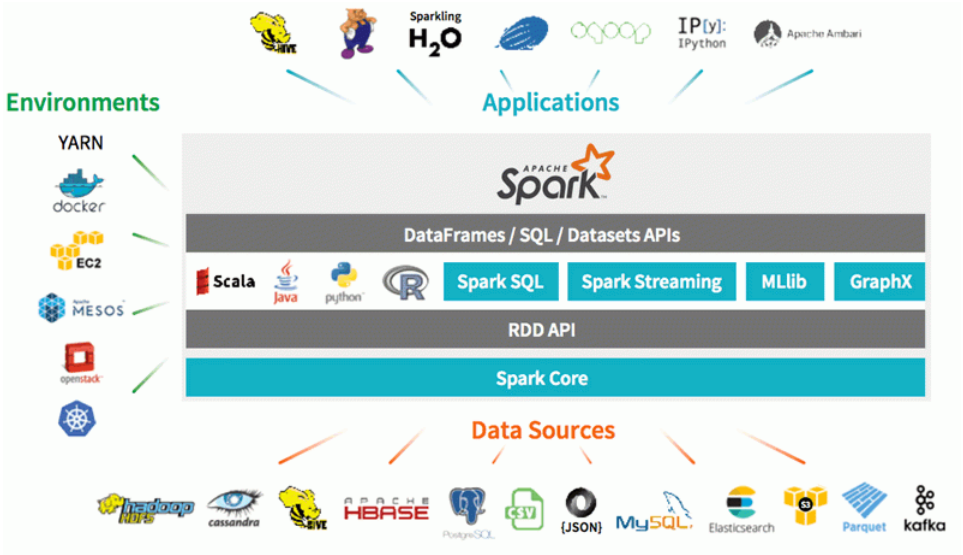


Figure 9: The world of Spark

Generally speaking, Spark is advance and highly capable upgrade to Hadoop aimed at enhancing Hadoop ability of cutting-edge analysis. Spark engine functions quite advance and different than Hadoop. Spark engine is developed for in-memory processing as well a disk-based processing. This in-memory processing capability makes it much faster than any traditional data processing engine [18].

Spark offers many distinct advantages over other distributed computing platforms, such as:

- A faster execution platform for both iterative machine learning and interactive data analysis.
- Single stack for batch processing, SQL queries, real-time stream processing, graph processing, and complex data analytics.
- Provides high-level API to develop a diverse range of distributed applications by hiding the complexities of distributed programming.
- Seamless support for various data sources such as RDBMS, HBase, Cassandra, Parquet, MongoDB, HDFS, Amazon S3, and so on[19].

Spark and distributed computing

The Reasons to build distributed systems abound. Oftentimes, the reason is the ability to tackle a problem so big that no individual computer could handle it at all, or at least, not in a reasonable amount of time [20].

2.2.4 Apache Cassandra

Cassandra first started as an incubation project at Apache in January 2009.[20] Apache Cassandra is free, open source, distributed data storage that differs sharply from relational database management system.

The Apache Cassandra database is the right choice when you need scalability and high availability without compromising performance. Linear scalability and proven fault-tolerance on commodity hardware or cloud infrastructure make it the perfect platform for mission-critical data[21]. Cassandra is based on a flexible schema.

Cassandra is a NoSQL data store based on the wide-column data model. NoSQL databases are increasingly replacing relational databases because of their inherent advantages of a flexible schema, ease of use, integration with Web applications, scalability, and integration with Apache Hadoop. Cassandra is ranked second among NoSQL databases. Cassandra is ranked 10th among all databases (relational or non-relational) [22].

Chapter 3

Analysis and design

This section analyzes the technologies involved, specifying in a concrete and detailed way what the program has to do, how it has to do it and see that they really adapt to the proposed requirements.

3.1 Analysis of requirements

The following section details the requirements of the application, which will help us understand, what is the problem or need to solve and how it will be addressed. That is, what the system has to do, or what need it will solve.

- A programming language compatible with the connection libraries between Spark and Cassandra is needed.

- The program has to use the Spark Parallelization tools to be run in distributed computing.

- A storage database is needed.

- A program that previously calculates the reference genome is needed, that is, to map all the sequences that make up the genome and that are stored. In this way to access these references when the algorithm is executed.

- A program is needed to create a genome hash. In other words, we need to storage the partitions of genome

- We need a program that by passing a Query by parameters, calculate the matching through the Smith-Waterman algorithm and give us a final result.

- Programs must be able to act in clusters.

3.2 Programming language

As mentioned above in other sections, Spark can be programmed in different APIs such as **Scala**, **Java**, **Python** and **R.** We discard R because it's specific to statistical analysis. We also discard Java because it's heavy language, that is, it's not a shell language or in other words doesn't support Read-Evaluate-Print-Loop.

Python and Scala are the best languages to program in this project. They are Object Oriented plus functional and have the same syntax and the most important thing is that both have libraries to connect with Cassandra easily. Python is very easy to understand, it's simpler and it's quick to find easy solutions on the net. Which also has a large number of libraries. In the other hand Scala is the native language of Apache Spark and runs 10 times faster than Python but is more complex to program and learn because it's abstract.

Choosing Scala seems the most logical solution, but it's required to have extensive knowledge in language. On the other hand, Python is a very well-known language for me. To carry out the complexity of this project, the best choice is to choose the dominant language. For these reasons we choose Python.

3.3 Technologies in project context

In this section, the technologies that the project implies are analyzed and justified. In each subsection of each technology we will show the structure that will have the programs that are going to be executed in Apache Spark and save in the Apache Cassandra database, also the structure of the database that Cassandra will have.

3.3.1 Spark

Apache Spark is a clear choice given that it is a widely used technology today and we need a scalable technology that works with large volumes of data and uses all the tools available to it, which makes it very powerful. Using Apache Spark we can execute and process the programs required for this project using the tools offered to us. The most relevant thing that it offers us and that we have used in the project is to calculate and process with Dataframes and RDDs to transform the data, group by, and map to exploit the distributed computing.

Next, we will present the programs that have been designed to obtain the final result of the sequence alignment. The process is divided into three parts, that is, into three programs.

Create genome hash

First of all we need to be able to access each genome sequence, so it is going to be mapped and create the complete hash.

In this first part of the process is the creation of hash. It will also be a program that can be run independently and without dependencies and we can divide the process into three sub-parts as shown in Figure 10.

First, there is an input parameter, a file that contains reference genome. Spark reads the file and sends each line to be processed by the Spark nodes.

Second, the nodes are responsible for separating into parts of size K and finding the positions where it has been found in the genome, we will call them offsets.

The last part is when everything is stored in Cassandra. For each record in the table

there is a sequence and the offsets. In this way, we have the entire genome mapped.

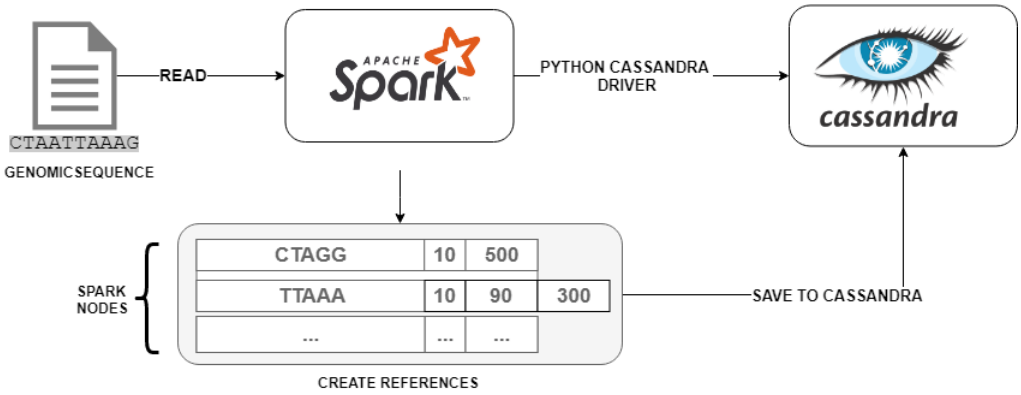


Figure 10: Hash creation scheme

Create genome reference in blocks

First of all we need to have the genome content saved to access it, so this program is created. In this second part of the process can be executed independently as well as the first.

As you can see in the figure 11, first, the genome file is read, then the genome reference is created and divided into blocks of size N, then stored in Cassandra. This is saved in this way to allow non-sequential indexed access.

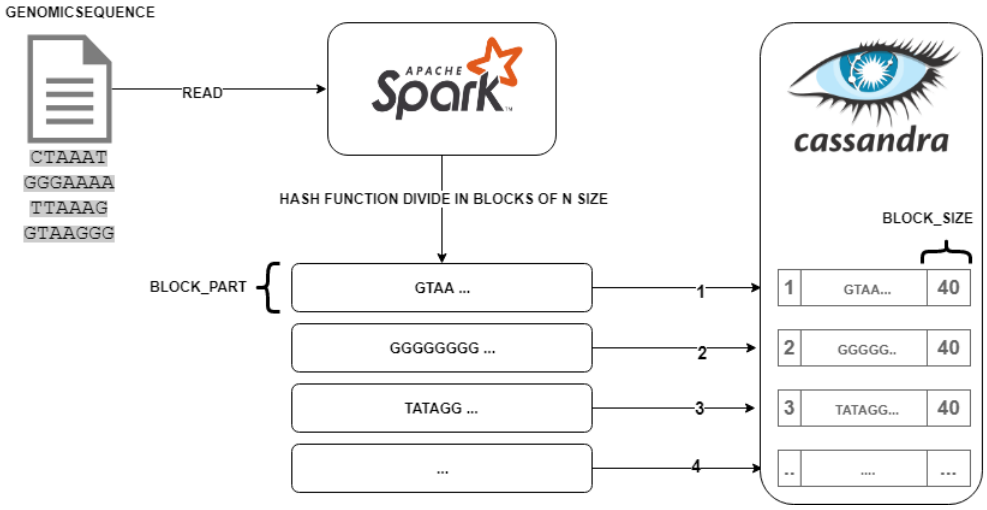


Figure 11: Reference content creation scheme

Process the Query and sequence alignment

There is a need to process the Queries and align them with the BLAST algorithm to yield the final result, so the following program is created.

In the third part of the process the previous two are needed to be able to execute.

The first step 1, as can be seen in the figure 12 is to process the Query to divide it. It is the same thing that is done in the program part of Create Hash 10. But now we would look in Cassandra if there are the sequences matching with saved in the first Create Hash program 10.

In this way, in step 2 of the figure 12 we apply the reduction only of the sequences found in step 1.

The final step would be to apply the alignment algorithm, during this process we will need to access the Create Reference program figure 11, to obtain the block where the sequence is located and to have the adjoining pieces to execute the algorithm. From here we get the final result of all the programs.

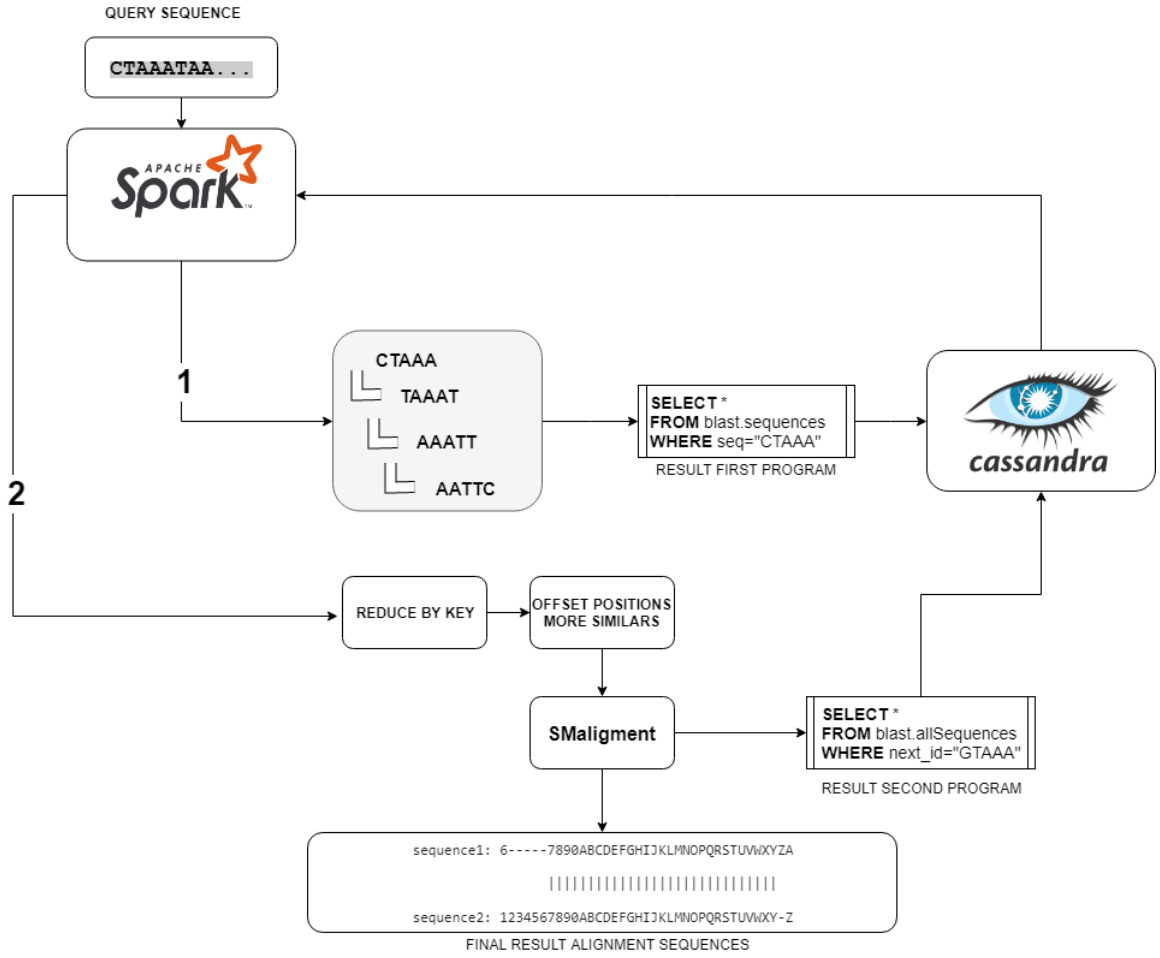


Figure 12: Query processing scheme

3.3.2 Cassandra

Apache Cassandra is chosen as the database manager. The other possibilities could have been, use the same HDFS file system that Hadoop carries or also use another external engine such as Mongo DB. But in the project definition, the database has to be structured and requires a high percentage of indexing at the table level, because it uses to make partitions within the database, so HDFS is left out and NoSQL databases is preferred. With NoSQL we will not have relations between tables, which makes the easy replication of the data in the different nodes of the cluster.

For these reasons Apache Cassandra is used as the database manager in this project. From spark we will connect with Cassandra through **DataStax Python Driver**

for Apache Cassandra. This will allow us to use the access tools directly from spark to Cassandra and vice versa, in a comfortable and easy way.

Cassandra will use it in our program to have the sequences and reference hash index stored persistently, because Spark has volatile memory and we only use the memory for data processing.

Database Schema

In Cassandra, we are presented with the following database schema in figure 13, within the keyspace defined in Cassandra, there are two tables described below:

Figure 13: Keyspace Blast

Mainly we will have the **sequences** table. Where the entire mapped reference genome is stored, that is, the reference hash. More explained, each line will contain a reference and a list of positions (offsets) from where that particular reference is found within the entire genome. It is structured in this way so that you can easily access the list of positions in each sequence and be able to execute the relevant calculations.

The main index is the field of **seq**. Because it is which will make all queries to Cassandra and as already mentioned above, indexing is very important, because that is where partitions between nodes and clusters will be made throughout the

development. For example, if more than one block needs to be consulted at the same time, it is not known which node each block can be in and what is attempted is that everything can be consulted from the same node or partition.

The other table that we will have created called **allSequences**, is to save the entire genome, but this time it will be partitioned by a block size K. In this way we will have in each line an identifier, the block part of size K and what size is each block, because for the final block of the genome, it will not match being the same size as all the others. The primary key **next_id** in allSequences is chosen for the following reasons. Has this structure, to easily find the sub sequence within the block of sequences:

- K = *bloc size (input parm)*
- $Candidate\_seq\_pos$ = *position of a matching sequence*

$$\frac{Candidate\_seq\_pos}{K} = next\_id$$

The **next_id** is the identifier we look for in the database, in this way it returns the block where the sequence is.

$$Candidate\_seq\_pos - K \cdot next\_id = localPosition$$

The **localPosition** is the subsequence position inside the block sequence.

Chapter 4

Implementation

This section explains in detail the development of the programs that we will process in Spark. Also the decisions taken throughout the implementation, the problems encountered and the solutions applied.

4.1 Init parameters

This subsection explains the initial parameters that the programs have, which can be changed for each program execution. They are the most relevant and help to understand the implementations.

Word size (key_size in program): Length of an exact sequence match, as start region for the final alignment. The default word size is 11. A BLAST search starts with finding a perfect sequence match of length given by word_size. This initial region of an exact sequence match is then extended in both direction allowing gaps and substitutions based on the scoring thresholds.

Changing the initial word-size can help to find more, but less accurate hits; or to limit the results to almost perfect hits.

- **Decreasing the word-size** will increase the number of detected homologous sequences, but hits can include alignments of higher fragmentation due to gaps and substitutions (example: search for homologous genes between distant species). The negative part is the consumption increase. The system will need

more resources to process.

- **Increasing the word-size** will give less hits as it requires a longer continuous regions of exact match. If the word-size is chosen to be almost the size of the query, BLAST will search for almost exact matches. The positive part is the lower consumption of resources because there are fewer partitions.

For short sequences, word-size must be less than half the query length, otherwise reliable hits can be missed.

Query: The query sequence is the input parameter that the user chooses to scan, process and evaluate the final hits. For example, it can be a protein.

BlockSize: The block size of the partitions. In case the reference genome file is of a large size, partitions would be used to make it more concurrent in the different nodes.

CreateWindowWithPartitions: As we have already mentioned in the BlockSize, if you want to activate the partitions or not.

4.2 Create hash

This program takes advantage of all the tools available from Spark, such as RDDs to parallelize data and Dataframes. Also the DataStax Driver to connect Spark to Cassandra. As described above, this program helps us to have the entire genome mapped in Cassandra, and later in other processes to be able to access through queries in Cassandra. This program is used to create the hash reference genome. It's divided into three phases.

In the listing 1 we can see the process. It is described below.

In phase one the genome file is imported in order to create the DataFrame in line 2.

In phase two, different transformations are applied in the Dataframe to be processed line by line. First, the current line with the next line is nested so as not to have dependencies. Second, an id is created to subtract the spaces that are created for each line break. All this happens between lines 5 - 25. In this way the lines are

already prepared to be sent to each spark node.

In In phase three, the nodes are responsible for processing their corresponding piece of keys. This would be on line 24. Finally everything is saved in Cassandra, each reference with the positions (offsets) where it was found.

```
1     #sqlc = SQLContext(sc)
2     reference_df = sqlc.createDataFrame(reference_rdd,["file_offset","line"])
3
4     # Delete first line (header)
5     header = reference_df.first()
6     header_size = len(header.line)+1
7     df = reference_df.filter(reference_df.file_offset!=0)
8
9     # Join line and prev_line.
10    if (CreateWindowWithPartitions and blockSize>0):
11        df = df.withColumn("block", (df.file_offset / blockSize).cast("int"))
12        my_window = Window.partitionBy("block").orderBy("file_offset")
13        print("Spark shuffle partitions:"
14            .format(sqlc.getConf("spark.sql.shuffle.partitions")))
15    else: # Without partitions
16        my_window = Window.partitionBy().orderBy("file_offset")
17    df1 = df.withColumn("line", F.upper(df.line))
18        .withColumn("id",F.row_number().over(my_window))
19    df2 = df1.withColumn("next_line", F.lag(df1.line,-1).over(my_window))
20    df3 = df2.withColumn("size", F.length(df2.line))
21        .withColumn("lines", F.when(F.isnull(df2.next_line), df2.line)
22                    .otherwise(F.concat(df2.line, df2.next_line)))
23        .withColumn("offset", (df2.file_offset-header_size)-df2.id)
24            .drop(df2.line).drop(df2.next_line)
25            .drop(df2.file_offset).drop(df2.id)
26
27     # Calculate keys
28    result = df3.rdd.flatMap(generateReferenceKeys)
```

Listing 1: Source code of create hash

4.3 Create reference

The following program was designed to keep the reference genome split into blocks of X size. For later, when the alignment algorithm is executed to be able to access the relevant block or blocks to find all the adjacent sequences and thus be able to

extend the sequences.

This program had to be done in simple Python language without being distributed. Concurrency problems were found. The more concurrence there was, the more problems arose. The main problem is that Spark works by lines and they are processed separately in parallel nodes. So to generate blocks of sequences joining lines is not useful.

Next we will briefly explain the steps. First in the program the genome file is read. As we can see in the listing 2, lines 4 to 13 are divided into blocks of size X, defined by input parameter. Finally it is saved in Cassandra through the Spark-Cassandra Driver connector of DataStax in the lines 17 to 20.

```
1
2   #Previous code ...
3
4    for line in f:
5             line = line.strip()
6             line = line.upper()
7             line = line.replace(" ", "")
8             if line:
9                 if len(line) % limit == 0:
10                    if line_save !=  "":
11                        line = line_save + line
12                        if len(line) % limit == 0:
13                            lines = [line[i:i+limit]
14                            for i in range(0, len(line), limit)]
15                            line_save =  ""
16                            for i in lines:
17                                session
18                                .execute("""INSERT INTO blast.allSequences
19                                          (next_id, part, size)
20                                                VALUES (%s, %s, %s)""",(k,i,len(i)))
21
22     # Later code ...
23
```

Listing 2: Source code of create references

4.4 Do query

In this program, it's explained the final procedures for processing the Query, it's compared with the keys that coincide with the Cassandra data already calculated in the previous programs. In this way obtain the final alignments with the Query.

In the listing 3. First in the function processQuery the Query is processed, it is partitioned to obtain the offsets and the offset increase for each Query is calculated and saved in a dictionary with the key-value key, offset From line 8 to 16. Then it is sent to parallelize the Query on line 19.

```
1  # Generate reference keys with the following tuples {key,offset}
2  # Does not store the key extension
3  futures = []
4  def processQuery(query):
5      LocalAlignments = []
6
7      # Calculate offset increment for each query and store it in a dictionary
8      offset_inc = dict()
9      query_len = len(query)
10     for k in range (0, query_len-key_size+1):
11         key = query[k:k+key_size]
12         offset_inc[key] = query_len - k - 1
13
14     listOffset = []
15     for k in offset_inc:
16         listOffset.append({'seq' : k , 'value' : offset_inc[k]})
17     rowsOffsetRDD = sc.parallelize(listOffset)
18
19     pairs = rowsOffsetRDD.flatMap(lambda r: execQueryParallel(r))
20
21     # Later code ...
```

Listing 3: Source code of do query - first step

Second in the listing 4, the dictionary is sent to the workers in the execQueryParallel function and each one processes its part. Workers make CQL queries to Cassandra and returns only the matching ones in lines from 4 to 8.

Third still in the listing 4, when workers return the values, they have to be grouped to see the totals. The reduceByKey(add) function is applied in line 10 to group and

add the results. At this point in line 11, the offsets are filtered on a threshold and the ones with the most matches are selected. In other words, only the pieces of the reference genome that exceed the minimum number of matching are those analyzed to be aligned.

```
1  def execQueryParallel(record):
2      futuresParallel = []
3
4      querySelect = "SELECT * FROM blast.sequences WHERE seq=%s"
5      session = cluster.connect()
6      resultSelect = session.execute(querySelect, [record['seq']])
7      session.shutdown()
8      cluster.shutdown()
9
10     result = pairs.reduceByKey(add)
11     result = result.filter(lambda x: x[1] > 5)
12
13     # Later code ...
```

Listing 4: Source code of do query - second step

The last step of the driver is to send the results filtered by the threshold to the workers through the workersAligment function that appears in listing 5.

Each worker will search for his block of sequences in the reference genome in Cassandra. If it coincides with two blocks, the two will be joined from line 5 to 13.

From here, the Smith-Waterman alignment algorithm is repeatedly applied with the SMalignment function to find the best alignment from line 19 to 30. Finally, the same nodes are filtered, only those that obtain a high matching, they will pass a second threshold and will be chosen to be shown in the final result.

```
def workersAligment(position):

    #Previous code ...

    while (candidate_seq_length > maxLenSeq):
        next_id = next_id +1
        querySelect = "SELECT * FROM blast.allSequences WHERE next_id=%s"
        cluster = Cluster(['192.168.1.1', '192.168.1.2', '192.168.1.3', '192.168.1.4',
                                                    '192.168.1.5', '192.168.1.6'])
        session = cluster.connect()
        resultSelect = session.execute(querySelect, [next_id])

        maxLenSeq = maxLenSeq + resultSelect[0].size
        sequence1 = sequence1 + resultSelect[0].part

    candidate_sequence = sequence1[:candidate_seq_length]

    i_start_indexs = []
    for i_start in range(15):
            _,_,score = SMalignment(candidate_sequence[i_start:],query_seq)
            i_start_indexs.append(score)

        i_start = max(range(len(i_start_indexs)), key=lambda x: i_start_indexs[x])
        i_end_indexs = []
        for i_end in range(1,16):
            _,_,score = SMalignment(candidate_sequence[:-i_end],query_seq)
            i_end_indexs.append(score)

        i_end = max(range(len(i_end_indexs)), key=lambda x: i_end_indexs[x])+1
        candidate_sequence = candidate_sequence[i_start:-i_end]
        align_seq1,align_seq2,align_score = SMalignment(candidate_sequence,query_seq)

        if align_score>0.7:
            print("FINAL find in "+str(candidate_seq_pos+i_start)+' ---> '+
                str(candidate_seq_pos+i_start+len(candidate_sequence)-1)+
                    ", align score: "+str(align_score))
            print("Final Align1 "+format(align_seq1))
            print("Final Align2 "+format(align_seq2))
            Display(align_seq1, align_seq2)

    # Later code ...


```

Listing 5: Source code of do query - third step

Chapter 5

Results and tests

In this chapter, the results obtained during the tests carried out on the programs will be presented, which have been executed in the **Cluster Babel** of the **University of Lleida**. Which has a specification of 6 nodes x 16 cores. These results will be presented below.

5.1 Program results

In this section I wanted to briefly show what the results of an execution of the three programs would be.

As we can see in the figure 14, the first line is the position where the sequence was found and the final score it obtained to be selected as a finalist. It is a score that has a 86% matching. The other parts of the image show the result of the alignment of sequence 1, which is the Query compared to another sequence 2 found in Cassandra.

```
FINAL find in 10207 ---> 10283, align score: 0.867469879518

sequence1: CCTAACCCTAACCCTAACCCTAACCCCTAACCCTAACCCTAAACCCTAAACC-----CTA
           |||||||||||||||||| |||||| |||||||||||||| | |||| ||     |||
sequence2: CCTAACCCTAACCCTAACC-TAACCC-TAACCCTAACCCTAACC-CTAACCCTAACCCTA

sequence1: AAACCCTAAACC-----CTAACCCTAACCCTAACCC-TAACCC
           || | |||| ||     |||||||||||||||||| ||||||
sequence2: AACC-CTAACCCTAACCCTAACCCTAACCCCTAACCC
```

Figure 14: Query alignment result

5.2 Performance tests

This section shows the results of the performance tests in the different programs. They are analyzed based on their scalability increasing to the input data sizes.

5.2.1 Times by size

Here we will count the time it takes for the two distributed programs. The objective of the test is the execution time grows linearly with the reference size.

The tests have been carried out with 6 nodes and 8 cores. What makes a total of 48 processors.

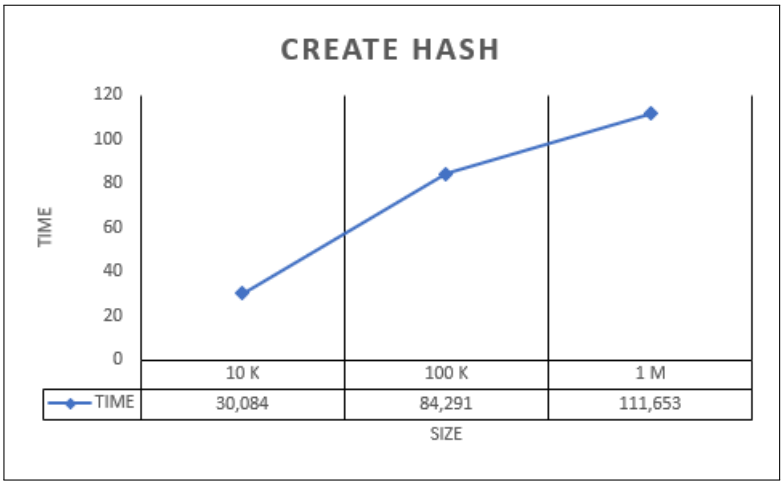


Figure 15: Results of execution tests of create hash

Size	Time (seconds)
10 K	30,084
100 K	84,291
1 M	111,653

Table 2: The final results of execution tests of Create Hash

As we can see in the figure 15 and in the table 2, the program is the creation of the reference hash and the time is counted based on the size of the genome input file. The conclusions that we can draw is that the program grows linearly in each execution of increase in size, which is a good sign.

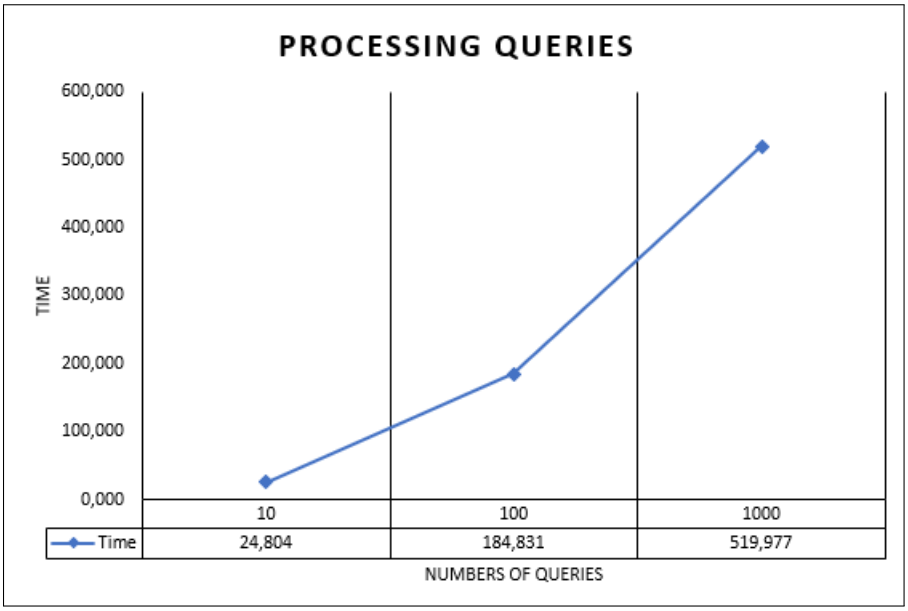


Figure 16: Results of execution tests of processing queries

Queries	Time (seconds)
10	24,804
100	184,831
1000	519,977

Table 3: The final results of execution tests of Process Query

In the case of the execution of the final Query processing and sequence alignment program, as we can see in the figure 16 and the table 3 the program of creating the reference will be analyzed differently, because in this program the file size does not matter so much, but it is more important The size of queries.Therefore, it has been thought of executing the program by launching and processing 10, 100, 1000 queries for each execution.

The conclusions that we can draw, is the program grows linearly in each execution of increase in size, but it seems to increase too much as the concurrence increases.

5.2.2 Speed-up

The speed-up is the Ratio between the serial and parallel execution time. We can see that in the following:

$$\frac{T1}{Tp} = Sp$$

For these tests, the execution was carried out with different executors and cores with the same genome input file, with a large size of 29MB.

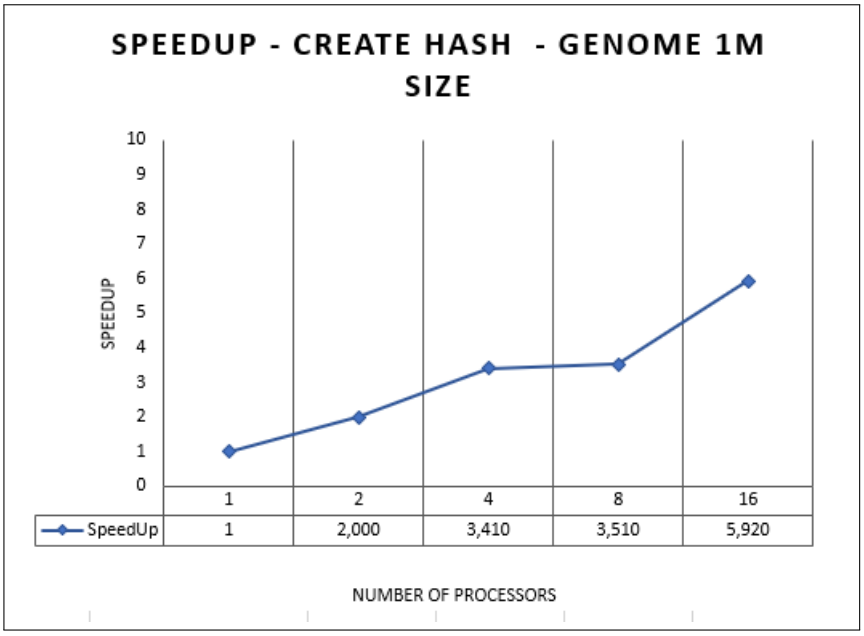


Figure 17: Results of execution tests of speedup of create hash

Number of Processors	SpeedUp
1	1
2	2,00
4	3,410
8	3,510
16	5,920

Table 4: The final results of execution tests of Speedup of create hash

In the case of the program of the creation of the hash reference, as we can see in the figure 17 and table 4.

The speedup is less than the number of processes, it is good for the scalability, which indicates the problem grows well when the problem size is increase. Although it reaches its peak to 4 processors, then go up little by little. This indicates that too much concurrence is also not good for an optimal result. Although in the 16 processors it it improves again.

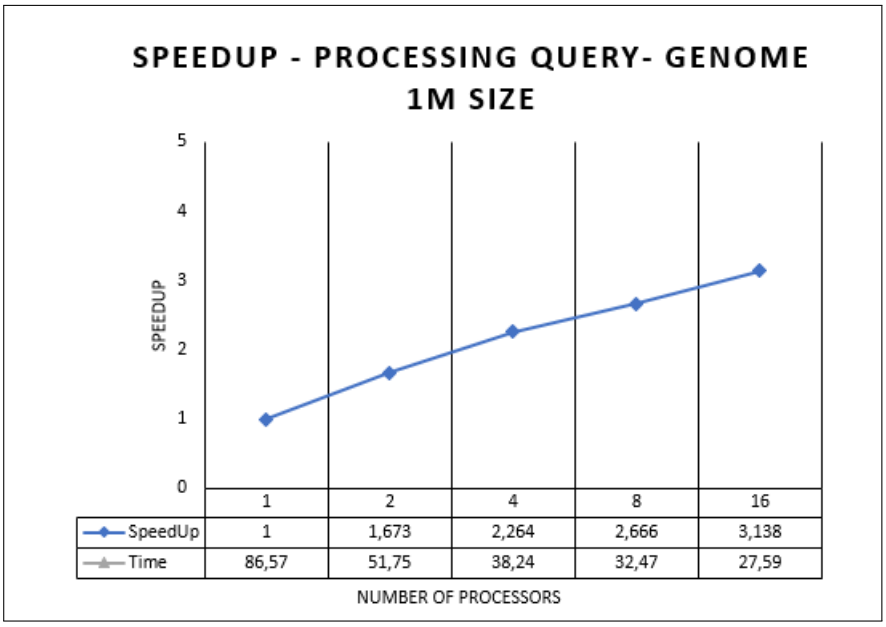


Figure 18: Results of execution tests of processing query

Number of Processors	SpeedUp
1	1
2	1,673
4	2,264
8	2,666
16	3,138

Table 5: The final results of speedup processing query

In the case of the program of the processing query, as we can see in the figure 18 and table 5

The speedup is less than the number of processors, remains linear and increases without peaks. That means scalability is good.

5.2.3 Efficiency

Ratio between the speedup factor and the number of processors. This is a measure of the cost-efficiency of computations.

$$\frac{1}{p} \leq E = \frac{S_p}{p} = \frac{T_1}{pT_p} \leq 1$$

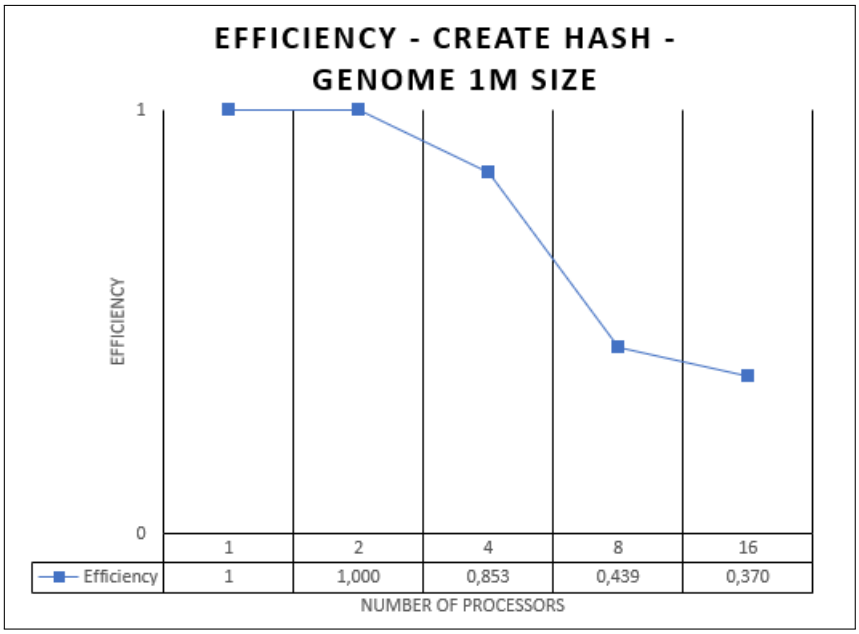


Figure 19: Efficiency of create hash

Number of Processors	Efficiency
1	1
2	2,00
4	3,410
8	3,510
16	5,920

Table 6: Efficiency of create hash

The efficiency of create hash in figure 19 and the table 6 The efficiency remains ideal with 2 processors, it's good at 4. From 4 to 8 the fall is much worse. But in 16 it seems to stabilize. It very efficient with few processors, when they increase they no longer but remains.

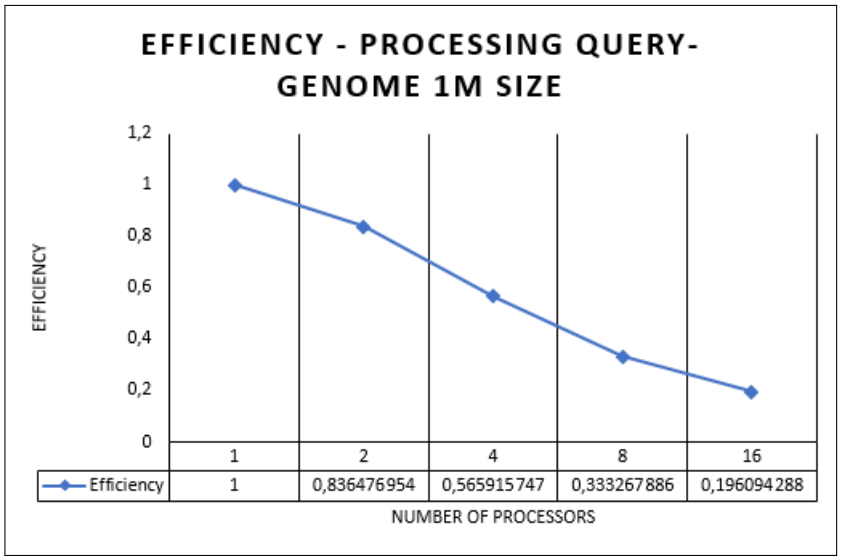


Figure 20: Results of efficiency processing query

Number of Processors	Efficiency
1	1
2	0,836476954
4	0,565915747
8	0,333267886
16	0,196094288

Table 7: Efficiency of processing query

The efficiency of create hash in figure 20 and the table 7 The efficiency remains good in 2 processors, it starts to be medium in 4. Then at 8 and 16 continues to decline. The program does not seem to scale as a larger number of queries have to be processed.

Chapter 6

Conclusions

First of all, I would like to say, I have learned more deeply about how Spark works, how to improve Python knowledge, how to execute more customizable processes in Cluster. Also how Spark works with Cassandra, the programs that are needed for communication between these two tools to work and learn the tools it provides. Actually this last point of the communication of Spark-Cassandra has been more difficult, since the functions were those carried by the installed connector and I think they are somewhat limited. Especially the part of call Cassandra, forces you to open a new connection for each process and that penalizes enough time to greatly increase concurrency.

Although much research has been done prior to implementation. The problems caused by making the processes concurrent, have led to the rethinking of the implementations and going in other ways throughout the development because they were not viable. That has made the implementation of the programs more extended.

On the objectives set, mainly to adapt to distributed computing the development of processing genomes and queries to perform the alignments of genetic sequences has been overcome. Not 100%, because there are processes that have had to be performed sequentially as is the case of the program that generates the genome partitioned in blocks. In addition, some problems also arise to process very large genomes, because Cassandra is not able to store all the positions (offsets) of the

sequence in a single line.

6.1 Future work plan

As we have said in the previous section, the program can still be further optimized and made more concurrent. Because in the tests carried out the results are also not that they are excessively good. There is work ahead.

The problem that you cannot save all the offsets on a single line in Cassandra would be a future job. Try to find another way to partition and use another index to index by sequence and block of sequence. Another problem is the one commented on the way Cassandra is called from Spark, opening each time connection for each node. It would be interesting to delve deeper into this topic to find other not so heavy ways.

List of source codes

List of Figures

List of Tables

Bibliography

[1] Macías, M. & Gómez, M. *Introducción a Apache Spark: para empezar a programar el big data* (Editorial UOC, 2015). URL https://ebookcentral.proquest.com/lib/inacapsp/detail.action?docID=4536453.

[2] Holmes, D. E. *Big Data: una breve introducción* (Antoni Bosch editor, 2018). URL https://ebookcentral.proquest.com.

[3] Hernández-Leal, E. J., Duque-Méndez, N. D. & Moreno-Cadavid, J. Big data: una exploración de investigaciones, tecnologías y casos de aplicación. *Revista Tecno Lógicas* **20**, 1 – 24 (2017). URL https://ezproxy.dnb-inacap.cl/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=fap&AN=123557860&lang=es&site=eds-live&scope=site.

[4] Kashyap, H., Ahmed, H. A., Hoque, N., Roy, S. & Bhattacharyya, D. K. Big data analytics in bioinformatics: A machine learning perspective (2015). 1506.05101.

[5] Melgarejo, S. Bioinformática: la salida profesional de futuro en medicina personalizada (2017).

[6] Langkafel, P. *Big Data in Medical Science and Healthcare Management : Diagnosis, Therapy, Side Effects.* (De Gruyter, 2016). URL https://ezproxy.dnb-inacap.cl/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=e000xww&AN=1107028&lang=es&site=eds-live&scope=site.

[7] Ferraro Petrillo, U., Sorella, M., Cattaneo, G., Giancarlo, R. & Rombo, S. E. Analyzing big datasets of genomic sequences: fast and scalable collection of k-mer statistics. *BMC Bioinformatics* **20**, 1 – 14 (2019). URL https://ezproxy.dnb-inacap.cl/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=asn&AN=135962793&lang=es&site=eds-live&scope=site.

[8] Aguilar-Bultet, L. & Falquet, L. Secuenciación y ensamblaje de novo de genomas bacterianos una alternativa para el estudio de nuevos patógenos . *Revista de Salud Animal* **37**, 125 – 132 (2015). URL http://scielo.sld.cu/scielo.php?script=sci_arttext&pid=S0253-570X2015000200008&nrm=iso.

[9] Núñez, M., Nguyen, N. T., Camacho, D. & Trawiński, B. Computational collective intelligence: 7th international conference, iccci 2015. *ICCCI* **2**, 1 – 14 (2015).

[10] González, G. Algoritmo blast. URL http://www.bioinformaticos.com.ar/algoritmo-blast/.

[11] Bio, C. Bioinformatics explained blast (2007). URL http://www.ccg.unam.mx/~vinuesa/tlem/pdfs/Bioinformatics_explained_BLAST.pdf.

[12] Agnellutti, C. *Big Data : An Exploration of Opportunities, Values, and Privacy Issues.* Internet Theory, Technology and Applications (Nova Science Publishers, Inc, 2014). URL https://ezproxy.dnb-inacap.cl/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=nlebk&AN=811106&lang=es&site=eds-live&scope=site.

[13] Manyika, J. *et al.* Big data: The next frontier for innovation, competition, and productivity. *Big Data: The Next Frontier for Innovation, Competition & Productivity* 1 – 143 (2011). URL https://ezproxy.dnb-inacap.cl/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=bsu&AN=76406289&lang=es&site=eds-live&scope=site.

[14] Apache hadoop. URL https://hadoop.apache.org/.

[15] Santos, P. R. d. l. Hadoop por dentro (ii): Hdfs y mapreduce - think big empresas (2019). URL https://empresas.blogthinkbig.com/hadoop-por-dentro-ii-hdfs-y-mapreduce/.

[16] Salloum, S., Dautov, R., Chen, X., Peng, P. X. & Huang, J. Z. Big data analytics on apache spark. *International Journal of Data Science and Analytics* **1**, 145–164 (2016). URL https://doi.org/10.1007/s41060-016-0027-9.

[17] IEEE. Performance prediction for apache spark platform. *2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems, High Performance Computing and Communications (HPCC), 2015 IEEE 7th International Symposium on Cyberspace Safety and Security (CSS), 2015 IEEE 12th International Conferen on Embedded Software and Systems (ICESS), 2015 IEEE 17th International Conference on* 166 (2015). URL https://ezproxy.dnb-inacap.cl/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=edseee&AN=edseee.7336160&lang=es&site=eds-live&scope=site.

[18] Shoro, A. d. G. & Soomro, T. R. Big data analysis: Ap spark perspective. *Global Journal of Computer Science and Technology C Software & Data Engineering* **15**, 7–11 (2015).

[19] Duvvuri, S. & Laxmikanth, V. *Spark for Data Science.* (Packt Publishing, 2016). URL https://ezproxy.dnb-inacap.cl/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=nlebk&AN=1364691&lang=es&site=eds-live&scope=site.

[20] Pierfederici, F. *Distributed Computing with Python.* Community Experience Distilled (Packt Publishing, 2016). URL https://ezproxy.dnb-inacap.cl/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=nlebk&AN=1220461&lang=es&site=eds-live&scope=site.

[21] Foundation, T. A. S. (2016). URL http://cassandra.apache.org/.

[22] Vohra, D. *NoSQL Web Development with Apache Cassandra.* (Cengage Learning PTR, 2015). URL https://ezproxy.dnb-inacap.cl/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=nlebk&AN=947932&lang=es&site=eds-live&scope=site.